Instituto Politécnico Nacional

Escuela Superior De Computo

Sistemas en Chip

Práctica 5

Nombre de los integrantes:

García Quiroz Gustavo Iván

Bejarano García Owen Uriel

Grupo: 7CV3

Nombre del Profesor: Miguel Ángel Castillo Martínez

Fecha De Entrega: 03/06/2025

1 Índice

2 Introducción

En esta práctica se implementará un sistema de adquisición y procesamiento de señales utilizando el microcontrolador ATmega328P (integrado en la tarjeta Arduino UNO), con el objetivo de leer una señal analógica proveniente de un potenciómetro, convertirla a formato digital mediante el ADC interno, y utilizar este valor para modular una señal PWM (Modulación por Ancho de Pulso) generada por el Timer0. Adicionalmente, se utilizará la comunicación UART para transmitir en tiempo real tanto el valor digital obtenido como el voltaje calculado, permitiendo su monitoreo desde un programa terminal.

A diferencia de las implementaciones comunes basadas en funciones de alto nivel como analogRead() y analogWrite(), en esta práctica se trabajará exclusivamente con manipulación directa de registros, sin el uso de la API de Arduino, así se tendrá un mayor control sobre los periféricos del microcontrolador y facilitará la comprensión de su funcionamiento interno.

Finalmente, para validar el comportamiento del sistema, se utilizará un osciloscopio digital para observar tanto la señal analógica del potenciómetro como la señal PWM generada, comprobando que el ciclo de trabajo se ajusta dinámicamente en función del valor del ADC. Esta práctica sienta las bases para el desarrollo de aplicaciones más complejas que requieren lectura de sensores y control de salidas mediante técnicas de modulación.

3 Objetivo General

Analizar el funcionamiento del Convertidor Analógico-Digital (ADC) del microcontrolador ATmega328P mediante la lectura de una señal variable generada por un potenciómetro, utilizando tanto la manipulación directa de registros como herramientas de visualización (Serial Plotter y osciloscopio).

3.1 Objetivos Específicos

- Implementar un circuito básico con Arduino UNO y un potenciómetro de 10kΩ para generar una señal analógica variable (0-5V).
- Configurar el ADC mediante manipulación de registros del ATmega328P (registros ADMUX, ADCSRA y ADC), evitando el uso de funciones predefinidas como analogRead().
- Visualizar los resultados digitales en el Serial Plotter de Arduino IDE, interpretando la relación entre el valor ADC (0-1023) y el voltaje analógico (0-5V).
- Validar las mediciones con un osciloscopio, comparando la señal analógica real (osciloscopio) con la digitalizada (ADC).
- Evaluar la precisión y limitaciones del ADC, identificando fuentes de error como ruido eléctrico o errores de cuantización.

4 Materiales y Equipo Utilizado

Componentes electrónicos

- 1 Arduino UNO (ATmega328P).
- 1 Protoboard y cables dupont.
- 1 Cable USB.
- 1 Potenciómetro.

Herramientas e instrumentos

- 1 Osciloscopio digital.
- 1 Computadora con Arduino IDE instalado.

Software

- Arduino IDE.

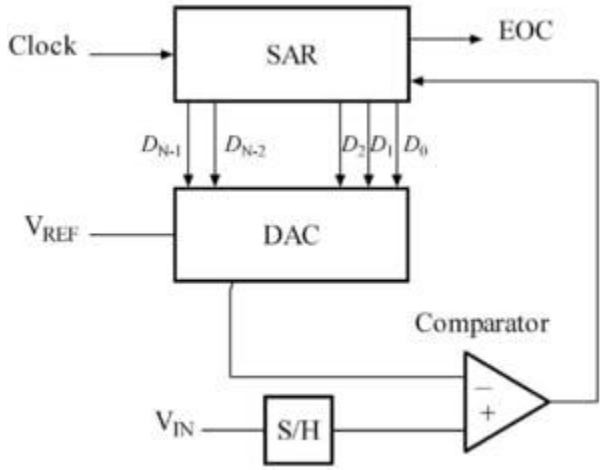
5 Marco Teórico

5.1 Convertidor Analógico-Digital (ADC)

El ADC del ATmega328P es un perifal que transforma señales analógicas (ej. voltaje de un potenciómetro) a valores digitales (10 bits, rango 0–1023). Sus aracterísticas principales son:

- **Resolución**: 10 bits (210=1024210=1024 niveles).
- **Voltaje de referencia**: Por defecto usa VCC=5V*VCC*=5*V* (configurable mediante registros ADMUX).
- **Tiempo de conversión**: Depende del *preescaler* (ej. 125 kHz con división por 128 en un Arduino a 16 MHz)[1].

Internamente opera mediante un método de aproximaciones sucesivas (SAR, Successive Approximation Register), el cual compara el voltaje analógico de entrada contra un voltaje generado internamente por un DAC (convertidor digital a analógico). El proceso comienza fijando el bit más significativo (MSB) en 1 y viendo si el voltaje del DAC excede la entrada, dependiendo del resultado, el bit se mantiene o se limpia, y el proceso continúa con el siguiente bit menos significativo, hasta completar los 10 bits. Este método permite una conversión rápida y precisa, ideal para microcontroladores de propósito general.



Desde su introducción en los microcontroladores AVR de 8 bits, como los de la familia ATmega, este tipo de ADC ha sido clave en el desarrollo de prototipos rápidos, sistemas embebidos de bajo costo y soluciones de IoT, manteniéndose vigente por su bajo consumo, integración y facilidad de configuración a nivel de registros.

5.2 Registros del ADC en ATmega328P

La configuración directa vía registros permite mayor control:

- ADMUX:
    - Bits REFS1:0: Seleccionan referencia de voltaje (ej. 01 para AVcc).
    - Bits MUX3:0: Eligen el canal ADC (ej. 0000 para ADC0/A0).
- ADCSRA:
    - Bit ADEN: Habilita el ADC.
    - Bits ADPS2:0: Ajustan el preescaler (ej. 111 para división por 128).
- ADC: Registro de 16 bits (10 bits útiles) que almacena el resultado[1][2].

5.3 Potenciómetro como Divisor de Voltaje

Un potenciómetro de 10kΩ actúa como un divisor resistivo variable:

- **Salida (wiper)**: Entre 0V (GND) y 5V (VCC*VCC*), proporcional al ángulo de giro.
- **Ecuación**: Vout=VCC·RwiperRtotal*V*out=*VCC*·*R*total*R*wiper.

6 Desarrollo de la Práctica

6.1 Montaje del circuito

Para comenzar la práctica, se conectó un potenciómetro lineal de 10kΩ a la tarjeta Arduino UNO, siguiendo esta configuración:

- Pin izquierdo del potenciómetro → GND de Arduino.
- Pin derecho → 5V de Arduino.
- Pin central (wiper) → Entrada analógica A0 (canal ADC0 del ATmega328P).

Además, se conectó la punta del osciloscopio al pin central del potenciómetro para visualizar la señal analógica de entrada, y a la salida digital del pin 6 (OC0A), donde se genera una señal PWM proporcional al valor leído por el ADC mediante el Timer0.

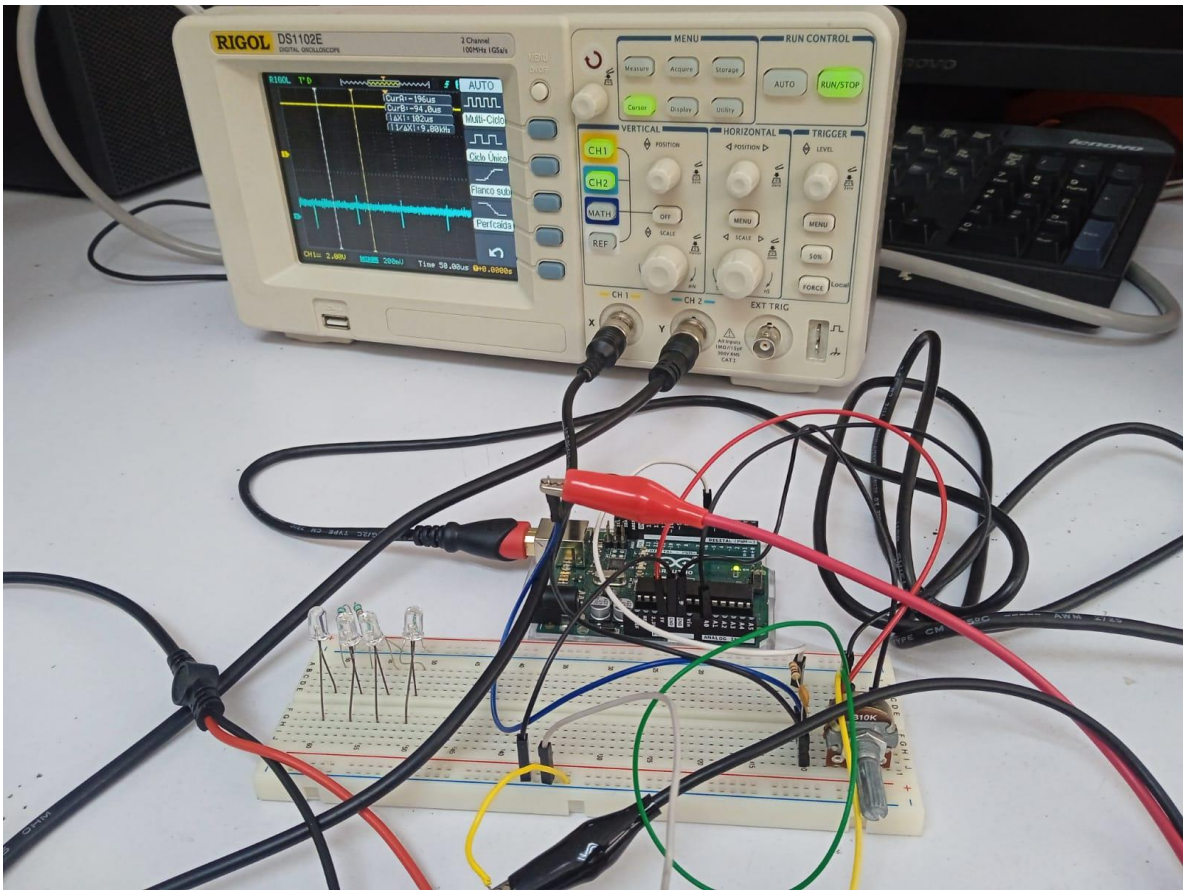


Figura 1

6.2 Configuración del código

Se programó el microcontrolador ATmega328P de la placa Arduino UNO utilizando únicamente manipulación directa de registros, sin emplear funciones de alto nivel como analogRead(), pinMode() o setup()/loop(). Los pasos clave de configuración fueron:

- Configuración del ADC (registro ADMUX): Se seleccionó la referencia de voltaje como AVcc con un capacitor externo opcional en AREF (REFS1:0 = 01). El canal de entrada utilizado fue el ADC0 (pin A0), seleccionando MUX3:0 = 0000.
- Ajuste del preescaler del ADC (registro ADCSRA): Se estableció un preescaler de 128 (ADPS2:0 = 111), lo cual permite que el ADC opere dentro del rango recomendado de frecuencia para obtener lecturas estables.
- Habilitación e inicio del ADC: Se habilitó el ADC (ADEN = 1) y se iniciaron las conversiones de forma continua controladas por software mediante la bandera ADSC.
- Configuración del Timer0 para salida PWM: El Timer0 fue configurado en modo Fast PWM utilizando el pin OC0A (pin digital 6 del Arduino UNO) como salida. El ciclo de trabajo (duty cycle) del PWM se actualizó continuamente en función del valor leído por el ADC.
- Configuración del UART: Se inicializó la comunicación serie mediante los registros UBRR0, UCSR0A/B/C, estableciendo una velocidad de 9600 baudios. A través del UART se envió el valor crudo del ADC junto con su correspondiente voltaje, permitiendo observar los cambios en tiempo real desde un monitor serie.

Código importante:

```
28 void adc_init(void) {
29     // AVcc como referencia, canal ADC0
30     ADMUX = (1 << REFS0);
31 
32     // Habilitar ADC y prescaler de 128
33     ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
34 }
```

Figura 2

Código completo:

```
1  #include <avr/io.h>
2  #include <util/delay.h>
3  #include <stdio.h>
4  
5  #define F_CPU 16000000UL
6  #define BAUD 9600
7  #define MYUBRR ((F_CPU / (16UL * BAUD)) - 1)
8  
9  void uart_init(unsigned int ubrr) {
10     /*Set baud rate */
11     UBRR0H = (unsigned char)(ubrr >> 8);
12     UBRR0L = (unsigned char)ubrr;
13     UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Habilita RX y TX
14     UCSR0C = (1 << USBS0) | (3 << UCSZ00); // 8 bits de datos, 1 bit de stop
15 }
16 
17 void uart_transmit(char data) {
18     while (!(UCSR0A & (1 << UDRE0))); // Esperar buffer libre
19     UDR0 = data;
20 }
21 
22 void uart_print(const char* str) {
23     while (*str) {
24         uart_transmit(*str++);
25     }
26 }
27 
28 void adc_init(void) {
29     // AVcc como referencia, canal ADC0
30     ADMUX = (1 << REFS0);
31 
32     // Habilitar ADC y prescaler de 128
33     ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
34 }
35 
```

```c
uint16_t adc_read(uint8_t channel) {
    ADMUX = (ADMUX & 0xF0) | (channel & 0x0F); // Seleccionar canal

    ADCSRA |= (1 << ADSC); // Iniciar conversión
    while (ADCSRA & (1 << ADSC)); // Esperar

    return ADC;
}

void pwm_timer0_init(void) {
    // Configurar OC0A (PD6) como salida
    DDRD |= (1 << PD6);

    // Fast PWM, 8-bit: WGM02:0 = 3, no invertido
    TCCR0A = (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);
    TCCR0B = (1 << CS01); // Prescaler 8 → ~7.8 kHz PWM
}

int main(void) {
    char buffer[32];

    adc_init();
    pwm_timer0_init();
    uart_init(MYUBRR);

    while (1) {
        uint16_t adc_val = adc_read(0); // Leer ADC0 (A0)
        uint8_t pwm_val = adc_val >> 2; // Convertir 10 bits a 8 bits

        OCR0A = pwm_val; // PWM en OC0A (PD6 / pin 6)

        // Calcular voltaje aproximado (en mV)
        uint16_t mv = (adc_val * 5000UL) / 1023;

        // Formatear cadena para imprimir
        snprintf(buffer, sizeof(buffer), "ADC=%u | V=%u.%03uV\r\n", adc_val, mv / 1000, mv % 1000);
        uart_print(buffer);

        _delay_ms(300);
    }
}
```

Figura 3

6.3 Visualización de datos por UART

Para visualizar los resultados de la conversión analógica-digital, se implementó una interfaz por UART mediante manipulación directa de registros del ATmega328P, sin utilizar la biblioteca Serial ni el Serial Plotter del Arduino IDE, los datos enviados al monitor serie incluyen:

1. Valor digital crudo del ADC (0–1023)
2. Voltaje calculado (0–5V), obtenido con la fórmula:

$$V = \frac{ADC \times 5}{1023}$$

6.4 Medición con osciloscopio

Para validar los resultados del sistema de adquisición y generación de señal, se utilizó un osciloscopio digital conectado tanto a la señal analógica del potenciómetro como a la salida PWM generada por el Timer0 del ATmega328P:

Configuración del osciloscopio

- Modo DC.
- Escala vertical: 2V.
- Escala horizontal: 50us.

Procedimiento

1. Se giró el potenciómetro lentamente y se observó el cambio en el voltaje. Observe además que el ciclo de trabajo de la señal mostrada correspondiente al pin OC0A varía en función del valor entregado por el ADC, que a su vez depende de la posición del potenciómetro.
2. Se comparó el valor mostrado en el osciloscopio con el voltaje calculado por el Arduino.

Figura 4. Salida de la terminal y osciloscopio cuando el potenciómetro está en posición media (≈2.5V)

Figura 5. Salida de la terminal y osciloscopio cuando el potenciómetro está en posición máxima (≈5V)

7 Resultados

7.1 Datos Obtenidos

Posición del Potenciómetro	Valor ADC (0-1023)	Voltaje Calculado (V)	Voltaje Osciloscopio (V)
Mínima (GND)	0	0.00	0.00
50% (Media)	508	2.48	~2.48
Máxima (5V)	1009	4.92	~4.92

Al analizar los datos podemos decir lo siguiente

- **Precisión del ADC:** Los valores del ADC y el osciloscopio coinciden con un error mínimo, lo que confirma la linealidad del potenciómetro y la correcta configuración del ADC.
- **Ruido en las mediciones**: Se observó un pequeño ruido en el osciloscopio (picos de ~0.4V), probablemente debido a interferencias en las conexiones.

8 Conclusiones

A través de esta práctica, se comprobó que el ADC del ATmega328P convierte de manera efectiva la señal analógica del potenciómetro en valores digitales, manteniendo una relación lineal entre el voltaje de entrada (0-5V) y el valor leído (0-1023), la concordancia entre las mediciones del osciloscopio (señal analógica) y los valores mostrados en el Serial Plotter (señal digitalizada) confirma que la configuración directa de los registros del ADC es precisa y confiable para aplicaciones que requieren un control más fino que el proporcionado por la función analogRead().

Además, la práctica demostró que el acceso a bajo nivel a los registros del microcontrolador (como ADMUX y ADCSRA) permite optimizar el uso del ADC, ajustando parámetros como la velocidad de muestreo (mediante el preescaler) o la referencia de voltaje, esto es esencial en proyectos donde se requiera mayor eficiencia o se deban integrar múltiples periféricos.

9 Referencias

[1] ATmega328P Datasheet, Microchip Technology Inc., 2020. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[2] Arduino, "AnalogRead Serial," Arduino Documentation, 2023. [Online]. Available: https://docs.arduino.cc/built-in-examples/analog/AnalogReadSerial

[3] J. M. Hughes, Arduino: A Technical Reference, 1st ed. O'Reilly Media, 2016, pp. 45-50.

[4] Texas Instruments, Understanding Data Converters, SLAA013, 1995. [Online]. Available: https://www.ti.com/lit/an/slaa013/slaa013.pdf